Neuro-Logic Lifelong Learning

Bowen He[*], **Xiaoan Xu, Alper Kamil Bozkurt, Vahid Tarokh, Juncheng Dong**[†]

Duke University

[*]Corresponding author:bowen.he@duke.edu
[†]Advising

Abstract

Solving Inductive Logic Programming (ILP) problems with neural networks is a key challenge in Neural-Symbolic Artificial Intelligence (AI). While most research has focused on designing novel network architectures for individual problems, less effort has been devoted to exploring new learning paradigms involving a sequence of problems. In this work, we investigate lifelong learning ILP, which leverages the compositional and transferable nature of logic rules for efficient learning of new problems. We introduce a compositional framework, demonstrating how logic rules acquired from earlier tasks can be efficiently reused in subsequent ones, leading to improved scalability and performance. We formalize our approach and empirically evaluate it on sequences of tasks. Experimental results validate the feasibility and advantages of this paradigm, opening new directions for continual learning in Neural-Symbolic AI.

Keywords: Neuro-Symbolic AI, ILP, Lifelong Learning

1 Introduction

Neuro-Symbolic Artificial Intelligence (Santoro et al. 2017; Manhaeve et al. 2018; Dai et al. 2019; d'Avila Garcez and Lamb 2020; Amizadeh et al. 2020) has emerged as a promising research direction that combines modern neural networks with classic symbolic methods, thereby leveraging the strengths of both. At a high level, neural networks offer the expressivity and end-to-end learning capabilities needed to tackle complex problems where traditional symbolic methods can fall short. Meanwhile, symbolic approaches contribute advantages such as explicit representation of knowledge and reasoning, in which neural networks often underperform (Valmeekam et al. 2023; Li et al. 2024; Sheth, Roy, and Gaur 2023). Although the term *Neuro-Symbolic Artificial Intelligence* spans a wide range of problems, paradigms, and methodologies, this work focuses specifically on tasks within the field of **inductive logic programming** (ILP) (Cropper and Dumančić 2022). Unlike standard logic programming, which draws conclusions from a given set of rules, ILP seeks to learn first-order logic rules that best explain observed examples from given relevant background knowledge.

To solve ILP problems using neural networks, researchers have introduced a variety of methods and compared them against traditional ILP solvers (Evans and Grefenstette 2018; Glanois et al. 2021; Payani and Fekri 2019; Dong et al. 2019; Badreddine et al. 2022; Sen et al. 2022; Zimmer et al. 2023), demonstrating that neural network-based methods are both robust to noisy data and more efficient for large-scale tasks.

However, most existing works primarily focus on designing new model architectures, dedicating relatively little attention to investigate learning paradigms beyond the conventional individual tasks setting. To this end, this work takes the first step to investigate the transferability of knowledge between ILP problems. Our insight is that ***logic rules, by their nature, are compositional and reusable***. Specifically, a logic rule learned from one task can be naturally reused for another task within the same domain. Moreover, cognitive scientists have argued that humans learn, think, and reason in a symbolic manner, i.e.,*"symbolic models of cognition were the dominant computational approaches of cognition"* (Castro et al. 2025; Besold and Kühnberger 2023). Thus, to achieve the remarkable capabilities of lifelong learning and meta-learning observed in human intelligence, we envision that the interplay of neural network and symbolic method presents a promising new direction for lifelong learning.

We instantiate the aforementioned insight by introducing a novel lifelong learning problem for ILP. We introduce a compositional structure for neural logic models and evaluate their performance across sequences of tasks. By leveraging rules acquired from previous tasks, the neural logic models achieve significantly improved learning efficiency on new tasks. In comparison to the existing works that primarily focus on the perspective of model parameter optimization—such as regularization-based (Kirkpatrick et al. 2017; Zenke, Poole, and Ganguli 2017), experience replay-based (Rolnick et al. 2019; Buzzega et al. 2020), and architecture-based approaches (Rusu et al. 2016; von Oswald et al. 2020; Li et al. 2019)—lifelong learning for logic rules requires identifying which rules are beneficial for reuse and efficiently constructing new rules based on those already acquired. We take the first step in demonstrating the feasibility of this direction, paving the way for future research. Our empirical results confirm the enhanced learning efficiency achieved through lifelong learning. Furthermore, by simply incorporating experience replay, the model effectively retains its performance across tasks. Additionally, in certain experiments, we observe a backward transfer effect, where training

on later tasks further improves performance on earlier tasks.

Contribution Statement. We summarize our contributions as follows:

- We formally introduce lifelong learning in ILP, framing it as a sequential optimization problem.

- We propose a neuro-symbolic approach that leverages the compositionality of logic rules to enable knowledge transfer across tasks.

- We validate on challenging logic reasoning tasks how *logic rule transfer improves learning efficiency* and the acquisition of a common knowledge base during sequential learning.

Manuscript Organization. We frist briefly review related works in Section 2. We introduce the definition of lifelong ILP problems in Section 3. We elaborate on our implementation in Section 4. After presenting our experiment results in Section 5, we conclude with a discussion for future directions.

2 Related Works

We provide review for both inductive logic programming, extending the discussion to more recent neural network based approcach, and lifelong learning that aims to build AI systems that accumulate knowledge in a seuquence of tasks.

2.1 Inductive Logic Programming

Inductive Logic Programming (ILP) (Cropper and Dumančić 2022) is a longstanding and still unresolved challenge in artificial intelligence, characterized by its aim to solve problems through logical reasoning. Unlike statistical machine learning, where predications are based on statistical inference, ILP relies on logic inference and learns a logic program that could be used further to solve problems. Its integration with reinforcement learning (RL) further leads to the field of relational RL (Džeroski, De Raedt, and Driessens 2001), where the policy comprises logical rules and decisions are made through logical inference. The goal of ILP is to design AI systems that not only solve problems but do so through logical reasoning, which recent literature shows is lacking in purely neural network-based approaches (Valmeekam et al. 2023; Li et al. 2024). This raises a critical question: can we develop neural-symbolic methods that leverage the scalability of neural networks while also solving tasks through sound logical reasoning? (Evans and Grefenstette 2018) made a pioneering step towards this objective by introducing ∂ILP, which integrates neural networks with inductive logic programming (ILP) and addresses 20 ILP tasks sourced from previous literature or designed by the authors themselves. They showed that, compared to traditional ILP methods, ∂ILP is more robust against mislabeled targets that typically impair the performance of conventional approaches, reflecting the generalization capabilities of neural networks. Furthermore, (Jiang and Luo 2019) extends ∂ILP to the reinforcement learning setting by incorporating logic predicates into the state and action spaces. They evaluate its performance on two tasks,

Blocksworld and Gridworld, comparing it against MLP neural networks. Their results demonstrate that MLPs are prone to failure on these tasks represented with logic predicates.

Neural logic machine (NLM) (Dong et al. 2019) represents another line of research that aims to design better neural network architectures for logic reasoning. They proposed a forward chaining approach to represent logical rules, effectively addressing the memory cost issues associated with handling a large number of objects, a challenge previously encountered by the ∂ILP method. Despite the fact that the logic rules learned by NLMs are not explicitly extractable for human readers, they continue to be the most widely used benchmark in further applications (Wang et al. 2025) and related works. Differentiable Logic Machines (Zimmer et al. 2023) builds upon NLM to replace MLPs used by NLM with soft logic operator, providing more interpretability in the cost more computational requirement.
(Campero et al. 2018) proposes to learning vector embeddings for both logic predicates and logic rules. It's further expended by (Glanois et al. 2021) using the forward-chaining perspective as in NLM.

Logic rules, by its nature, provide the property of compositionality, a new logic rule could be always built by composing rules that have been acquired before (Lin et al. 2014). Moreover, logic rules that are learned in a task could be naturally transferred to be used in another task from the same domain. This ability of knowledge transfer and lifelong learning has been considered as essential for human-like AI (Lake et al. 2016). Building on this insight, we investigate the lifelong learning ability of neural logic programming.

2.2 Lifelong Learning

Lifelong learning or continual learning, has been proposed as a research direction for developing artificial intelligence systems capable of learning a sequence of tasks continuously (Wang et al. 2024). Ideally, a lifelong learning agent should achieve both *forward transfer*, where knowledge from earlier tasks benefits subsequent ones, and *backward transfer*, where learning newer tasks enhances performance on previous ones. At the very least, it should mitigate catastrophic forgetting, a phenomenon where learning later tasks causes the model to lose knowledge acquired from earlier tasks.

Classic lifelong learning methods typically fall into four categories: *(i) Regularization-based approaches*, which constrain parameter updates within a certain range to mitigate forgetting (Kirkpatrick et al. 2017; Zenke, Poole, and Ganguli 2017; Li and Hoiem 2017)); *(ii) Replay-based approaches*, which store or generate data samples from past tasks and replay them during training on newer tasks (Rebuffi et al. 2017; Shin et al. 2017; Lopez-Paz and Ranzato 2017; Chaudhry et al. 2018); *(iii) Optimization-based methods*, which manipulate gradients to preserve previously acquired knowledge (Zeng et al. 2019; Farajtabar et al. 2020; Saha, Garg, and Roy 2021)); and *(iv) Architecture-based methods*, which expand or reconfigure model architectures to accommodate new tasks or transfer knowledge (Rusu et al. 2016; Yoon et al. 2017). In all cases, these methods are mainly rooted in neural networks and can be broadly

categorized as strategies for optimizing model parameters or architectures. In contrast, approaching lifelong learning from a neuro-symbolic perspective offers additional opportunities by leveraging the properties of symbolic methods. Our work, therefore, falls within this emerging paradigm.

One work highly relevant to our study is (Mendez 2022). They propose incorporating compositionality into lifelong training, enabling new tasks to benefit from previously learned neural modules. However, their formulation remains within the domain of pure neural networks and can be categorized as an architecture-based method for lifelong learning. In contrast, we emphasize that logic rules inherently provide compositionality, which can be leveraged for learning. Thus, our work serves as a strong instance of lifelong learning with compositionality, facilitating the systematic reuse and adaptation of learned knowledge across tasks. Another work that explores lifelong learning from a neuro-symbolic perspective is (Marconato et al. 2023). However, their focus is on extracting reusable concepts from sub-symbolic inputs while preventing reasoning shortcuts that could lead to incorrect symbolic knowledge. In contrast, our work centers on the transfer of logic rules, where learning and reusing these rules play a fundamental role.

3 Neural Logic Lifelong Learning

3.1 Problem Formulation

We introduce our problem definition of ILP. We first define *objects* and their corresponding *types* within the domain of interest. Next, we define *predicates* and *operations*. Finally, we frame ILP as an optimization problem.

Object Sets. We denote the set of objects in a given domain as $\mathcal{O}$, while another set $\Lambda = \{\lambda_1, \lambda_2, ..., \lambda_n\}$ defines all possible types that these objects can take, namely, $\forall o \in \mathcal{O}, type(o) \in \Lambda$. A partition of $\mathcal{O}$ is thus induced by grouping objects of the same type into a subset. Formally, we partition the set of objects $\mathcal{O}$ into subsets $\{\mathcal{O}_\lambda\}_{\lambda \in \Lambda}$, where

- Each subset is nonempty: $\mathcal{O}_\lambda \neq \varnothing$;
- Subsets are pairwise disjoint: $\mathcal{O}_\lambda \cap \mathcal{O}_{\lambda'} = \varnothing$ for all $\lambda \neq \lambda'$;
- Their union forms the entire set: $\cup_{\lambda \in \Lambda} \mathcal{O}_\lambda = \mathcal{O}$;
- Objects within a single subset share the same type: $\forall \lambda \in \Lambda, \forall o_i, o_j \in \mathcal{O}_\lambda, type(o_i) = type(o_j)$.

Predicates. Next we define predicates, which are the cores of ILP programs. Consider $N \in \mathbb{Z}_{\geq 0}$. A N-ary predicate $\mathcal{P}$ is a binary-valued function $\mathcal{P} : \overline{\mathcal{O}}(\mathcal{P}) \to \{0, 1\}$ where $\mathcal{O}(\mathcal{P})$ is the Cartesian product of N elements, each of which arbitrarily selected from $\{\mathcal{O}_{\lambda_1}, \mathcal{O}_{\lambda_2}, \ldots, \mathcal{O}_{\lambda_n}\}$, that is,

$$\mathcal{O}(\mathcal{P}) = \mathcal{O}_{\lambda_{i_1}} \times \mathcal{O}_{\lambda_{i_2}} \times \cdots \times \mathcal{O}_{\lambda_{i_N}},$$

where $\lambda_{i_k} \in \Lambda$ for all $1 \leq k \leq N$. Any set of N-ary predicates $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m\}$ can be composed to form a new predicate $\widetilde{\mathcal{P}} = F(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m)$ with a *logic rule F*.

Consider an example with the object set $\mathcal{O} = \{o_1, o_2\}$ where $type(o_1) = type(o_2)$. We define two 1-ary predicates $\mathcal{P}_1$ and $\mathcal{P}_2$ over $\mathcal{O}$ as $\mathcal{P}_1(o_1) = 0, \mathcal{P}_1(o_2) = 1$; $\mathcal{P}_2(o_1) = 1, \mathcal{P}_2(o_2) = 1$. With $\mathcal{P}_1$ and $\mathcal{P}_2$ defined above,

we can compose a new predicate $\mathcal{P}_3$ defined as $\mathcal{P}_3(X) \leftarrow \mathcal{P}_1(X) \wedge \mathcal{P}_2(X)$. Here, the applied logic rule $F(\mathcal{P}_1, \mathcal{P}_2)$ is $\mathcal{P}_1(X) \wedge \mathcal{P}_2(X)$, leading to the valuation of $\mathcal{P}_3$ as

$$\mathcal{P}_3(o_1) = 0, \mathcal{P}_3(o_2) = 1.$$

ILP Problems. Based on the definitions above, we are now ready to define ILPs. An ILP takes a set of *background knowledge B* as input. Here, B is a set of m predicates $B = \{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_m\}$. Given a known target predicate $\mathcal{P}^\star$, the goal of an ILP program is to learn a logic rule $F^\star$ such that $F^\star(B) = \mathcal{P}^\star$. We follow the conventions to assume complete knowledge of B, that is, each $\mathcal{P}_i$ from B can be valuated on all possible inputs $o \in \mathcal{O}(\mathcal{P}_i)$, where $\mathcal{O}(\mathcal{P}_i)$ is the input space of $\mathcal{P}_i$. We denote the space of all logic rules in consideration as $\mathcal{F}$. An instance of the ILP problem is defined by a tuple $(\mathcal{O}, \Lambda, B, E)$, where $(\mathcal{O}, \Lambda, B)$ follow the previous definitions and $E = \{(o, y = \mathcal{P}^\star(o)) | o \in \mathcal{O}(\mathcal{P}^\star), y \in \{0, 1\}\}$ describes the knowledge about the target predicate $\mathcal{P}^\star$. The solution to the problem is a logic rule $\widehat{F}$ such that

$$\widehat{F} \in \arg\max_{F \in \mathcal{F}} \sum_{o, y \in E} \mathbb{1}(\widehat{\mathcal{P}}(o) = y) \quad \text{s.t.} \quad \widehat{\mathcal{P}} = F(B) \tag{1}$$

3.2 Lifelong Learning ILP

Now we define the *lifelong learning ILP problem* (L2ILP). Consider a sequence of target predicates $\{\mathcal{P}_1^\star, \mathcal{P}_2^\star \ldots\}$ sharing the same background knowledge B. Intuitively, this means that we observe a common set of foundational knowledge from which different target conclusions are to be inferred. For example, in a medical diagnosis setting, the background knowledge B could include general medical facts such as symptoms and their possible causes. Each target predicate $\mathcal{P}_i^\star$ could represent a different diagnostic task, such as fever, infection or other specific disease. We note that a naive approach to the above problem is to independently search for the optimal logic rule $\widehat{F}_t$ for each target predicate $\mathcal{P}_t^\star$. Yet, this method would largely overlook the potentially shared structure of the predicates among target predicates and result in significant computational inefficiency. To this end, the goal of L2ILP is to efficiently find the logic rule $\widehat{F}_t$ for target predicate $\mathcal{P}_t^\star$, using knowledge of the previously learned logic rules $\{\widehat{F}_1, \ldots, \widehat{F}_{t-1}\}$ for composing the previous target predicates $\{\mathcal{P}_1^\star, \ldots, \mathcal{P}_{t-1}^\star\}$. While there may exist multiple approaches for this purpose, ***we propose to utilize the compositionality of logic rules and predicates by reusing the intermediate predicates composed during the learning of previous target predicates***, thereby efficiently composing new predicates.

Motivating Example. Consider an example of learning two target predicates on a graph reasoning task. A graph is described by a binary predicate $IsConnected(X, Y)$, additionally, several unary predicates are used to describe the colors of the nodes, such as $Red(X)$, $Yellow(X)$, $Blue(X)$, etc. A target predicate $AdjacentToRed(X)$ could be defined from a rule

$$AdjacentToRed(X) \leftarrow \exists Y IsConnected(X, Y) \wedge Red(Y)$$

Another target predicate $MultipleRed(X)$ could be similarly defined as

$$MultiRed(X) \leftarrow \exists Y \exists Z IsConnected(X, Y) \wedge Red(Y)) \wedge \\ IsConnected(X, Z) \wedge Red(Z)) \wedge \\ Is\_Not(Y, Z)$$

Learning these two predicates separately is computationally wasteful because the two rules share a common structure. Following this observation, we propose to solve the optimization problem through the reuse of logic rules across predicate functions, thus achieving forward transfer of knowledge required by lifelong learning.

Problem Formulation. Specifically, recall that $\mathcal{F}$ is the set of all possible logic rules in considerable. The goal of L2ILP to find a *shared knowledge base* $B_S \subset H(B)$ where $H(B) = \{F(B) | F \in \mathcal{F}\}$ is the set of all possible predicates that can composed from the background knowledge B. At time step t where the goal is to find a logic rule $\widehat{F}_t$ for $\mathcal{P}_t^\star$, we can *jointly* find B_S and $\widehat{F}_t$ through the following optimization problem,

$$\arg\max_{B_S, \widehat{F}_t} \sum_{t'=1}^{t} \sum_{o, y \in E_i} \mathbb{1}(\widehat{P}_{t'}(o) = y) \\ \text{s.t.} \quad \widehat{P}_{t'} = \widehat{F}_{t'}(B_S), \quad t' \in \{1, \ldots, t-1\}; \\ \widehat{P}_t = \widehat{F}_t(B_S), \quad \widehat{F}_t \in \widetilde{\mathcal{F}}_t, \quad (2)$$

where $\widehat{F}_{t'}$ for $\{1, \ldots, t-1\}$ are the learned logic rules for the target predicates previous to t. Problem (2) tries to identify a shared knowledge base B_S that can be *(i)* used by previously learned logical rules $\widehat{F}_{t'}$ for previous target predicts and *(ii)* used to find the logical rule $\widehat{F}_t$ for the current target predicate $\mathcal{P}_t^\star$. Notably, since L2ILP uses a shared knowledge base B_S to employ the compositionality of logic rules for efficient learning, the search space for $\widehat{F}_t$ (i.e., $\widetilde{\mathcal{F}}_t$) can be chosen to be a much smaller space than the space of all possible logic rules in consideration $\mathcal{F}$, i.e, $|\widetilde{\mathcal{F}}_t| \ll |\mathcal{F}|$. ***This can significantly increase the efficiency of learning***.

4 Compositional Neuro-Logic Model

Knowledge Base. We implement L2ILP using Neural Logic Machine (NLM) (Dong et al. 2019). NLM chains together a sequence of logic layers (i.e., neural network layers with predicates as inputs and outputs), where each layer can be viewed as learning logical rules over the immediate input predicates. In our formulation, each NLM layer can be interpreted as a search space for logical rules, denoted as $\mathcal{F}_{\mathrm{NLM}}$. Rules in $\mathcal{F}_{\mathrm{NLM}}$ are applied to the input predicates B to construct the output predicate space, i.e., $H_{\mathrm{NLM}}(B) = \{F(B) : F \in \mathcal{F}_{\mathrm{NLM}}\}$. Note that here the search space represented a NLM layer $\mathcal{F}_{\mathrm{NLM}}$ is much smaller than the whole search space $\mathcal{F}$. While one layer of NLM only generates target predicates with limited variation, chaining multiple layers lead to a complex and expressive target predicate space. Specifically, the target predicate space with n NLM layers can be defined recursively as

$$H_{\mathrm{NLM}}^n(B) = \{F(B \cup H_{\mathrm{NLM}}^{n-1}(B)) \mid F \in \mathcal{F}\}, \\ \text{where} \quad H_{\mathrm{NLM}}^1(B) = H_{\mathrm{NLM}}(B)$$

to represent the predicate space induced by iteratively applying logic rules from $\mathcal{F}_{\mathrm{NLM}}$.

We choose to construct the shared knowledge base $B_S \subset \bigcup_{i=1}^{n} H_{\mathrm{NLM}}^i(B)$ with the union of target predicates of increasing complexity $i \in \{1, \ldots, n\}$ where n is a hyperparameter to control the trade-off between the model complexity and target predicate expressiveness. In particular, by constructing the knowledge base with predicates of varying complexity, we achieve fine-grained predicate composition, reminiscent of the success of multi-scale representation-learning methods in the field of computer vision (Fan et al. 2021). Figure 1 illustrates this concept, where the knowledge base B_S is constructed by incorporating NLM layers of varying depths.
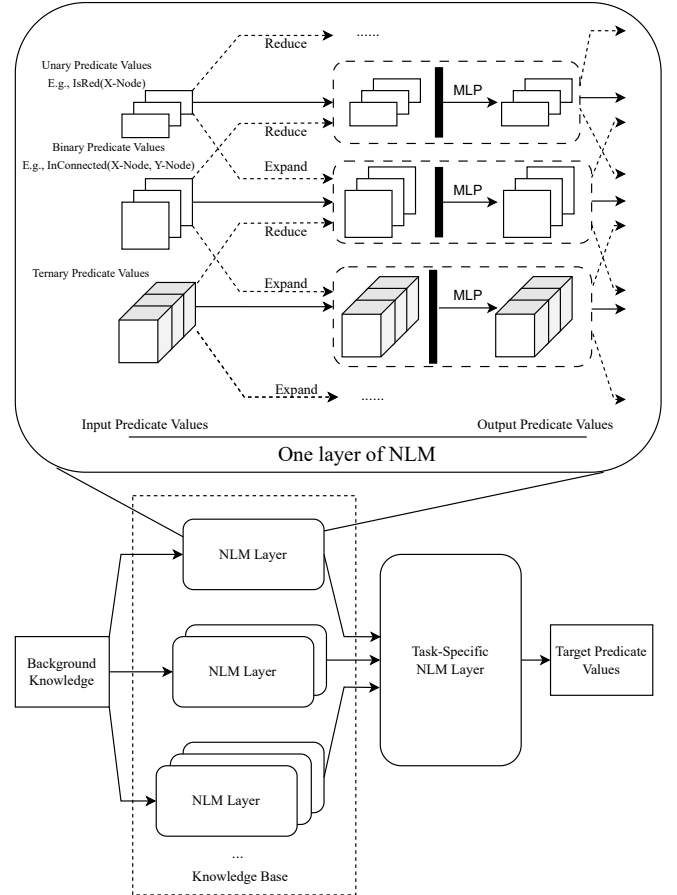

Figure 1: An illustration of Compositional Logic Model. Compositional Logic Model takes object properties and relations as input and outputs relations of the objects.

Task Specific Module. With the shared knowledge base B_S, the task specific module $\widetilde{\mathcal{F}}_i$ for $i \in \{1, ..., t\}$ is also a NLM layer to take as input all predicates from the knowledge base B_S and compose the target predicates for each corresponding task $\mathcal{P}_i^\star$. Figure 1 illustrates this concept, showing how a NLM layer utilizes predicates from the knowledge base to compose the target predicates for each task.

Training Protocol. We facilitate the transfer of the knowledge base across tasks while ensuring that task-specific modules remain distinct and independent from one another. This means that *the knowledge base is reused from task to task*, while task-specific modules are initialized randomly and trained from scratch.

5 Experiments

We note that the transfer of logic rules is beneficial to both supervised learning setting (i.e., ILP) and reinforcement learning setting (i.e., Relational RL). To this end, we present experimental results for both settings to comprehensively demonstrate the value of L2ILP. For supervised learning, we build on insights from the experiments conducted by (Dong et al. 2019), (Zimmer et al. 2023), (Glanois et al. 2021), and (Li et al. 2024), proposing task sequences for ILP across three domains: **arithmetic**, **tree**, and **graph**. For reinforcement learning, PDDLGym (Silver and Chitnis 2020) serves as an off-the-shelf tool in which the state and action spaces are represented using logical predicates. We therefore select **BlocksWorld** from PDDLGym as the testbed, as it is a commonly used benchmark for complex logic reasoning tasks (Džeroski, De Raedt, and Driessens 2001; Glanois et al. 2021; Valmeekam et al. 2023). ***We provide the code we used for experiments in the supplementary material.***

5.1 Forward Transfer of Logic Rules

ILP Experiments. The key question regarding the proposed approach is whether the transfer of logical rules is genuinely beneficial. We address this by comparing the learning curves of lifelong learning with those of models trained on tasks individually, while keeping model architectures the same. We provide a detailed description of the sequences of target predicates for each domain in the appendix, and present here the designs for two specific domains: Graph and Tree.

In the Graph domain, $IsConnected(X\text{-}Node, Y\text{-}Node)$ and $IsRed(X\text{-}Node)$, fully describe all possible graphs. The first predicate defines the connectivity between nodes, while the second specifies node properties—in this case, the color of the nodes. The four target predicates are learned sequentially in the following order:

- $AdjacentToRed(X\text{-}Node)$,
- $ExactConnectivity2(X\text{-}Node, Y\text{-}Node)$,
- $ExactConnectivity2Red(X\text{-}Node)$,
- $ExactConnectivity2MultipleRed(X\text{-}Node)$.

The first target predicate determines whether a given node has at least one neighboring node that is red. The second predicate identifies whether the shortest path between two nodes consists of exactly two edges. Building upon this, $ExactConnectivity2Red(X\text{-}Node)$ refines the notion of connectivity by verifying whether a node at an exact distance of two from the query node is red. Finally, $ExactConnectivity2MultipleRed(X\text{-}Node)$ extends this concept by determining whether there exist multiple such red nodes at the specified distance. We design the tasks in a

way that ensures relevance and allows them to share common structures when learning logical rules.

In the Tree domain, $IsParent(X\text{-}Node, Y\text{-}Node)$ is sufficient to specify the structure of any tree. We further define four target predicates to be learned sequentially, following the order below:

- $IsRoot(X\text{-}Node)$
- $HasOddEdges(X\text{-}Node, Y\text{-}Node)$
- $HasEvenEdges(X\text{-}Node, Y\text{-}Node)$
- $IsAncestor(X\text{-}Node, Y\text{-}Node)$

The first predicate identifies the root node of the tree and serves as the foundational predicate. The second and the third predicates determine whether two nodes in the tree are connected by an odd or even number of edges, respectively. Finally, $IsAncestor(X\text{-}Node, Y\text{-}Node)$ checks whether one node is an ancestor of another, based on their relative depths in the tree.

(a) Training dynamics for Arithmetic

(b) Training dynamics for Tree

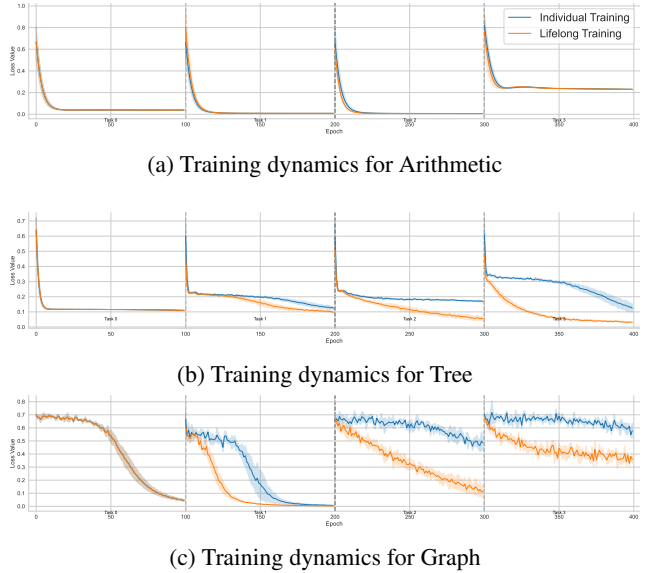(c) Training dynamics for Graph

Figure 2: Epoch training dynamics for individual learning and lifelong learning

Figure 2 illustrates the training dynamics for each task across all domains. For each task, we conducted experiments with four different seeds and plotted the mean, along with the standard deviation for each data point. From top to bottom, the domains are ordered as arithmetic, tree, and graph, while from left to right, the plots are arranged sequentially for tasks 0 to 3. The tasks in the arithmetic domain are relatively simple, allowing both individual and lifelong learning to converge to optimal performance within a short period. However, we observe that the loss curves in lifelong learning decrease more rapidly, as indicated by a small but noticeable gap. In contrast, for the tree and graph domains, the gaps become more pronounced, with no overlap observed between the paired curves. Notably, for the curves corresponding to the first task across all domains, lifelong learning completely overlaps with individual learning. This is expected, as the
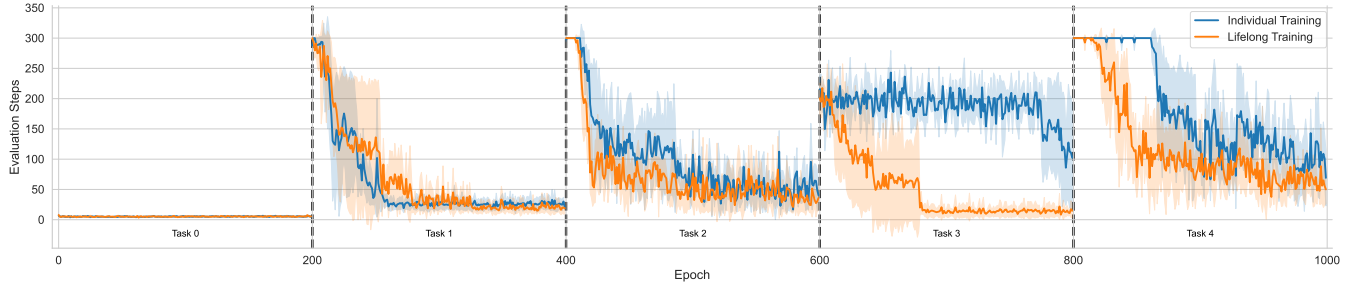
Figure 3: Evaluation steps for BlocksWorld tasks

first task in lifelong learning does not incorporate any previously acquired knowledge, making it equivalent to its individual learning counterpart.

This result shows that by leveraging the knowledge base acquired in earlier tasks, we could achieve higher learning efficiency on subsequence tasks, demonstrating the ***forward transfer*** effect that is essential in lifelong learning setting.

Relational RL Experiments. For the BlocksWorld task from PDDLGym, Table 1 summarizes the predicates used to describe the state and action spaces. Since the number of ground predicates depends on the number of objects in a task, the total number of possible actions can be as high as $2x^2 + 2x$, where x represents the number of blocks. However, not all of these actions are valid in every state. Additionally, the number of possible states grows factorially with the number of blocks in the environment, making the task increasingly complex as more blocks are added. We note that previous works (Jiang and Luo 2019; Zimmer et al. 2023; Valmeekam et al. 2023) typically use **5** blocks, whereas we extend our experiments to **6** blocks. Furthermore, We set the tasks to follow a sparse reward scheme, where a penalty of -0.1 is given for every step taken, while a reward of 100 is granted upon achieving the desired block configuration. To define the sequence of tasks for BlocksWorld, we assign different desired configurations to each task, progressively increasing the difficulty by making the target configurations more challenging to reach. Refer to Appendix for a detailed discussion.

Large state and action spaces pose significant challenges for exploration to RL agents, particularly when rewards are sparse and only provided upon goal completion. As a result, our experiments show that training tasks individually fails to yield sufficient exploration to positive rewards, let alone facilitate the learning of any meaningful policies. To address this issue, we adopt an **offline reinforcement learning** approach, where a replay buffer is collected in advance to ensure that both lifelong learning and individual training have access to data of the same quality. In BlocksWorld tasks, we use a planner to generate optimal actions at each step while incorporating random exploration to ensure adequate coverage of the state and action spaces. Specifically, we set the exploration rate to **0.8** and collect **50,000** transitions for each task.

Figure 3 illustrates the evaluation steps for each task, as the number of steps directly reflects the quality of the learned policy. We ran each experiment using four different seeds and evaluated the policy periodically. For tasks 0–2, lifelong training does not show superior performance compared to individual training, as these tasks are relatively simple. However, for the more challenging tasks 3 and 4, clear gaps emerge: lifelong training converges much faster, while individual training occasionally fails to converge to the optimal policy. This result highlights that ***forward transfer*** is not only observed in the relatively straightforward supervised learning setting but also extends to the more complex reinforcement learning paradigm, where effective knowledge transfer accelerates policy learning in later tasks.

5.2 Forgetting of Logic Rules

Forgetting Experiments. Another fundamental challenge in lifelong learning is assessing the extent to which a model forgets previously acquired knowledge (i.e. catastrophic forgetting) and identifying effective strategies to mitigate this issue. In our case, the knowledge base is continuously updated, which may cause task-specific modules for earlier tasks to fail, as they rely on outdated knowledge representations. To investigate this phenomenon, we track the loss values throughout the entire training process and visualize the corresponding curves for each task. Specifically, in the supervised learning setting, we adhere to the training protocol described in the previous section while recording the tested loss values for each task at every epoch whenever feasible. Figure 4 and Figure 5 illustrate the results for the graph

State Space		
Predicates	**Arity**	**Description**
$On(X\text{-}Block, Y\text{-}Block)$	2	Block X is on Block Y
$OnTable(X\text{-}Block)$	1	Block X is on the table
$Clear(X\text{-}Block)$	1	Block X could be picked up
$HandEmpty()$	0	Hand is empty
$HandFull()$	0	Hand is full
$Holding(X\text{-}Block)$	1	Hand is holding Block X
Action Space		
Predicates	**Arity**	**Description**
$PickUp(X\text{-}Block)$	1	Pick up Block X from the table
$PutDown(X\text{-}Block)$	1	Put down Block X onto the table
$Stack(X\text{-}Block, Y\text{-}Block)$	2	Stack Block X onto Block Y
$Unstack(X\text{-}Block, Y\text{-}Block)$	2	Unstack Block X from Block Y

Table 1: State and action spaces for BlocksWorld from PDDLGym

and tree domains, respectively, with each plot representing tasks 0 through 3. For results on arithemetic, please refer to appendix for a detailed description.
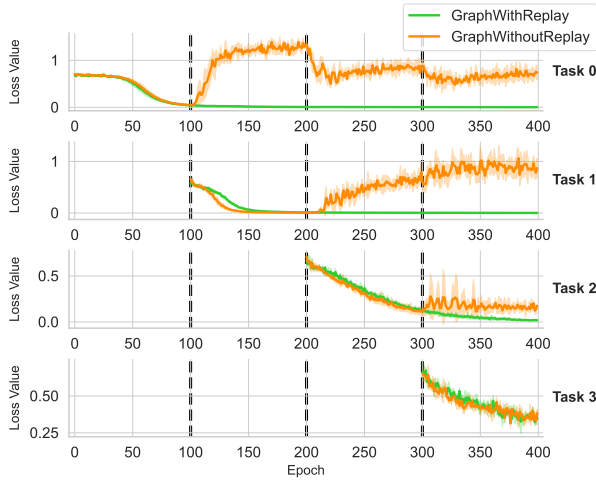

Figure 4: Training dynamics of Graph for each task

In particular, the orange curves illustrate the loss values as described above. As expected, the loss values for earlier tasks increase as training progresses on later tasks, indicating the occurrence of catastrophic forgetting in the model.

Replay Experiments. A straightforward approach to addressing catastrophic forgetting is to replay experience from earlier tasks while training on later ones. We adopt this strategy by replaying data from all previously learned tasks when training on a new task. This approach is particularly beneficial in our case, as we aim to build a shared knowledge base that can be effectively leveraged across multiple tasks. To ensure a fair comparison, we align the curves by recording each data point in terms of the training epoch of the current task. This alignment allows us to directly compare the performance of models trained with and without experience replay, providing deeper insights into its effectiveness in mitigating forgetting and preserving previously acquired knowledge.

The green curves in Figure 4 and Figure 5 illustrate the loss values when experience replay is applied. As the results indicate, during the training phase for each individual task, replaying experience does not significantly interfere with the learning of the current task. The loss curves for training without replay closely align with those for training with replay, indicating that incorporating past experiences does not disrupt the optimization of the current task. More importantly, experience replay effectively mitigates catastrophic forgetting, as evidenced by the green curves maintaining low loss values throughout the entire training process. This suggests that the model successfully constructs retains previously acquired knowledge, leading to the formation of a robust and reusable knowledge base across all tasks.

Surprisingly, we also observe a ***backward transfer*** effect in our experiments. The first plot in Figure 5 corresponds to the loss curve for task 0 in the tree domain. Notably, the
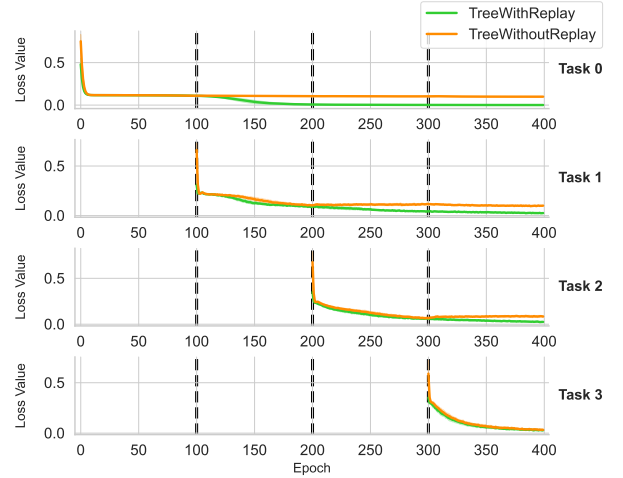

Figure 5: Training dynamics of Tree for each task

loss curve plateaus before training on task 1. However, once training on task 1 begins, the loss for task 0 further decreases to zero, indicating that learning task 1 enhances the model's performance on task 0. This suggests the potential of L2ILP that ***training on the current task may further improves the performance on previous tasks, as a better knowledge base is acquired during the sequential training***.

6 Conclusion

Neuro-Symbolic Artificial Intelligence introduces a new research paradigm by integrating neural networks with symbolic methods, which were traditionally studied as separate approaches. This integration opens new research opportunities by enabling the formulation of novel problems and providing solutions to challenges that were previously difficult to address using either approach alone. In this work, we take a step toward studying the lifelong learning problem in this domain and demonstrate that by leveraging the compositionality and transferability of logic rules, it becomes straightforward to construct models that achieve higher learning efficiency on later tasks while preserving performance on earlier ones. However, this problem remains far from solved. As discussed, a key challenge is how to efficiently construct logic rules that are meaningful for tasks within a given domain and how to systematically generate new rules from an evolving knowledge base. Our work represents a small step in this direction, demonstrating its feasibility but leaving many open opportunities for future research. We hope this study inspires further research into more effective methods for representing and constructing logic rules—particularly those that support the dynamic addition and removal of rules from a knowledge base—ultimately enabling more efficient and scalable neuro-symbolic lifelong learning systems.

References

Amizadeh, S.; Palangi, H.; Polozov, A.; Huang, Y.; and Koishida, K. 2020. Neuro-symbolic visual reasoning: Disentangling. In *International Conference on Machine Learning*, 279–290. Pmlr.

Badreddine, S.; d'Avila Garcez, A.; Serafini, L.; and Spranger, M. 2022. Logic Tensor Networks. *Artificial Intelligence*, 303: 103649.

Besold, T. R.; and Kühnberger, K.-U. 2023. *Symbolic and Hybrid Models of Cognition*, 139–172. Cambridge Handbooks in Psychology. Cambridge University Press.

Buzzega, P.; Boschini, M.; Porrello, A.; Abati, D.; and Calderara, S. 2020. Dark experience for general continual learning: a strong, simple baseline. In *Advances in Neural Information Processing Systems*, volume 33, 15920–15930.

Campero, A.; Pareja, A.; Klinger, T.; Tenenbaum, J.; and Riedel, S. 2018. Logical rule induction and theory learning using neural theorem proving. *arXiv preprint arXiv:1809.02193*.

Castro, P. S.; Tomasev, N.; Anand, A.; Sharma, N.; Mohanta, R.; Dev, A.; Perlin, K.; Jain, S.; Levin, K.; Éltető, N.; Dabney, W.; Novikov, A.; Turner, G. C.; Eckstein, M. K.; Daw, N. D.; Miller, K. J.; and Stachenfeld, K. L. 2025. Discovering Symbolic Cognitive Models from Human and Animal Behavior. *bioRxiv*.

Chaudhry, A.; Ranzato, M.; Rohrbach, M.; and Elhoseiny, M. 2018. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*.

Cropper, A.; and Dumančić, S. 2022. Inductive logic programming at 30: a new introduction. arXiv:2008.07912.

Dai, W.-Z.; Xu, Q.; Yu, Y.; and Zhou, Z.-H. 2019. Bridging machine learning and logical reasoning by abductive learning. *Advances in Neural Information Processing Systems*, 32.

d'Avila Garcez, A.; and Lamb, L. C. 2020. Neurosymbolic AI: The 3rd Wave. arXiv:2012.05876.

Dong, H.; Mao, J.; Lin, T.; Wang, C.; Li, L.; and Zhou, D. 2019. Neural Logic Machines. arXiv:1904.11694.

Džeroski, S.; De Raedt, L.; and Driessens, K. 2001. Relational reinforcement learning. *Machine learning*, 43: 7–52.

Evans, R.; and Grefenstette, E. 2018. Learning Explanatory Rules from Noisy Data. arXiv:1711.04574.

Fan, H.; Xiong, B.; Mangalam, K.; Li, Y.; Yan, Z.; Malik, J.; and Feichtenhofer, C. 2021. Multiscale Vision Transformers. arXiv:2104.11227.

Farajtabar, M.; Azizan, N.; Mott, A.; and Li, A. 2020. Orthogonal gradient descent for continual learning. In *International conference on artificial intelligence and statistics*, 3762–3773. PMLR.

Glanois, C.; Feng, X.; Jiang, Z.; Weng, P.; Zimmer, M.; Li, D.; and Liu, W. 2021. Neuro-Symbolic Hierarchical Rule Induction. arXiv:2112.13418.

Jiang, Z.; and Luo, S. 2019. Neural Logic Reinforcement Learning. arXiv:1904.10729.

Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A. A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13): 3521–3526.

Lake, B. M.; Ullman, T. D.; Tenenbaum, J. B.; and Gershman, S. J. 2016. Building Machines That Learn and Think Like People. arXiv:1604.00289.

Li, X.; Zhou, Y.; Wu, T.; Socher, R.; and Xiong, C. 2019. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In *International Conference on Machine Learning*, 3925–3934. PMLR.

Li, Z.; Cao, Y.; Xu, X.; Jiang, J.; Liu, X.; Teo, Y. S.; wei Lin, S.; and Liu, Y. 2024. LLMs for Relational Reasoning: How Far are We? arXiv:2401.09042.

Li, Z.; and Hoiem, D. 2017. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12): 2935–2947.

Lin, D.; Dechter, E.; Ellis, K.; Tenenbaum, J.; and Muggleton, S. 2014. Bias reformulation for one-shot function induction. In *ECAI 2014*, 525–530. IOS Press.

Lopez-Paz, D.; and Ranzato, M. 2017. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30.

Manhaeve, R.; Dumančić, S.; Kimmig, A.; Demeester, T.; and Raedt, L. D. 2018. DeepProbLog: Neural Probabilistic Logic Programming. arXiv:1805.10872.

Marconato, E.; Bontempo, G.; Ficarra, E.; Calderara, S.; Passerini, A.; and Teso, S. 2023. Neuro-symbolic continual learning: Knowledge, reasoning shortcuts and concept rehearsal. *arXiv preprint arXiv:2302.01242*.

Mendez, J. A. 2022. Lifelong Machine Learning of Functionally Compositional Structures. arXiv:2207.12256.

Payani, A.; and Fekri, F. 2019. Inductive Logic Programming via Differentiable Deep Neural Logic Networks. arXiv:1906.03523.

Rebuffi, S.-A.; Kolesnikov, A.; Sperl, G.; and Lampert, C. H. 2017. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2001–2010.

Rolnick, D.; Ahuja, A.; Schwarz, J.; Lillicrap, T.; and Wayne, G. 2019. Experience replay for continual learning. In *Advances in Neural Information Processing Systems*, 350–360.

Rusu, A. A.; Rabinowitz, N. C.; Desjardins, G.; Soyer, H.; Kirkpatrick, J.; Kavukcuoglu, K.; Pascanu, R.; and Hadsell, R. 2016. Progressive neural networks. In *arXiv preprint arXiv:1606.04671*.

Saha, G.; Garg, I.; and Roy, K. 2021. Gradient projection memory for continual learning. *arXiv preprint arXiv:2103.09762*.

Santoro, A.; Raposo, D.; Barrett, D. G.; Malinowski, M.; Pascanu, R.; Battaglia, P.; and Lillicrap, T. 2017. A simple neural network module for relational reasoning. *Advances in neural information processing systems*, 30.

Sen, P.; de Carvalho, B. W.; Riegel, R.; and Gray, A. 2022. Neuro-symbolic inductive logic programming with logical neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, 8212–8219.

Sheth, A.; Roy, K.; and Gaur, M. 2023. Neurosymbolic AI – Why, What, and How. arXiv:2305.00813.

Shin, H.; Lee, J. K.; Kim, J.; and Kim, J. 2017. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30.

Silver, T.; and Chitnis, R. 2020. PDDLGym: Gym Environments from PDDL Problems. arXiv:2002.06432.

Valmeekam, K.; Marquez, M.; Sreedharan, S.; and Kambhampati, S. 2023. On the Planning Abilities of Large Language Models : A Critical Investigation. arXiv:2305.15771.

von Oswald, J.; Henning, C.; Sacramento, J.; and Grewe, B. F. 2020. Continual learning with hypernetworks. In *International Conference on Learning Representations*.

Wang, C.; Ji, K.; Geng, J.; Ren, Z.; Fu, T.; Yang, F.; Guo, Y.; He, H.; Chen, X.; Zhan, Z.; Du, Q.; Su, S.; Li, B.; Qiu, Y.; Du, Y.; Li, Q.; Yang, Y.; Lin, X.; and Zhao, Z. 2025. Imperative Learning: A Self-supervised Neuro-Symbolic Learning Framework for Robot Autonomy. arXiv:2406.16087.

Wang, L.; Zhang, X.; Su, H.; and Zhu, J. 2024. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Yoon, J.; Yang, E.; Lee, J.; and Hwang, S. J. 2017. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*.

Zeng, G.; Chen, Y.; Cui, B.; and Yu, S. 2019. Continual learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, 1(8): 364–372.

Zenke, F.; Poole, B.; and Ganguli, S. 2017. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, 3987–3995. PMLR.

Zimmer, M.; Feng, X.; Glanois, C.; Jiang, Z.; Zhang, J.; Weng, P.; Li, D.; Hao, J.; and Liu, W. 2023. Differentiable Logic Machines. arXiv:2102.11529.

A Design for the Supervised Learning Tasks

A.1 Arithmetic

Input Predicates:

$$Zero(X\text{-}Number)$$ True if number X is zero

$$Succ(X\text{-}Number, Y\text{-}Number)$$ True if Y is the successor of X

Target Predicates:

1. $Plus(X\text{-}Number, Y\text{-}Number, Z\text{-}Number)$ True if $X + Y = Z$
2. $Times(X\text{-}Number, Y\text{-}Number, Z\text{-}Number)$ True if $X \times Y = Z$
3. $Division(X\text{-}Number, Y\text{-}Number, Z\text{-}Number)$ True if $X \div Y = Z$ (integer division)
4. $NoRemainder(X\text{-}Number, Y\text{-}Number)$ True if X is divisible by Y with no remainder

A.2 Tree

Input Predicates:

$$IsParent(X\text{-}Node, Y\text{-}Node)$$ True if X is the parent of Y

Target Predicates:

1. $IsRoot(X\text{-}Node)$ True if X is the root node
2. $HasOddEdges(X\text{-}Node, Y\text{-}Node)$ True if the path from X to Y has an odd number of edges
3. $HasEvenEdges(X\text{-}Node, Y\text{-}Node)$ True if the path from X to Y has an even number of edges
4. $IsAncestor(X\text{-}Node, Y\text{-}Node)$ True if X is an ancestor of Y

A.3 Graph

Input Predicates:

$$IsConnected(X\text{-}Node, Y\text{-}Node)$$ True if there is an edge between X and Y

$$IsRed(X\text{-}Node)$$ True if node X is red

Target Predicates:

1. $AdjacentToRed(X\text{-}Node)$ True if node X is connected to a red node
2. $ExactConnectivity2(X\text{-}Node, Y\text{-}Node)$ True if there is a path of exactly 2 edges from X to Y
3. $ExactConnectivity2Red(X\text{-}Node)$ True if X is connected via exactly 2 edges to a red node
4. $ExactConnectivity2MultipleRed(X\text{-}Node)$ True if X is connected via exactly 2 edges to multiple red nodes

B Design for the reinforcement learning tasks

The sequence of tasks in reinforcement learning varies based on their target configurations. For each task, we define specific configurations without considering the order of the blocks and sample a target configuration at the beginning of each episode. For example, if a task's target configuration requires three blocks to be on the table, two blocks to be stacked on others, and one block to be freely placed anywhere, we first generate possible configurations that meet this requirement and then sample a target configuration accordingly. This task design provides the flexibility to configure the target configuration space, allowing us to adjust the difficulty of the tasks as needed.

Thus, we define the five tasks in the following order:

- 5 blocks are on table, 1 block is free
- 3 blocks on table, 2 blocks are stacked, 1 block is free
- 2 blocks on table, 1 block is stacked, 1 block is free
- 4 blocks on table, 1 block is stacked, 1 block is free
- 1 block is on table, 4 blocks are stacked, 1 block is free

The order of the tasks are determined by the performance of the model trained on those tasks individually.

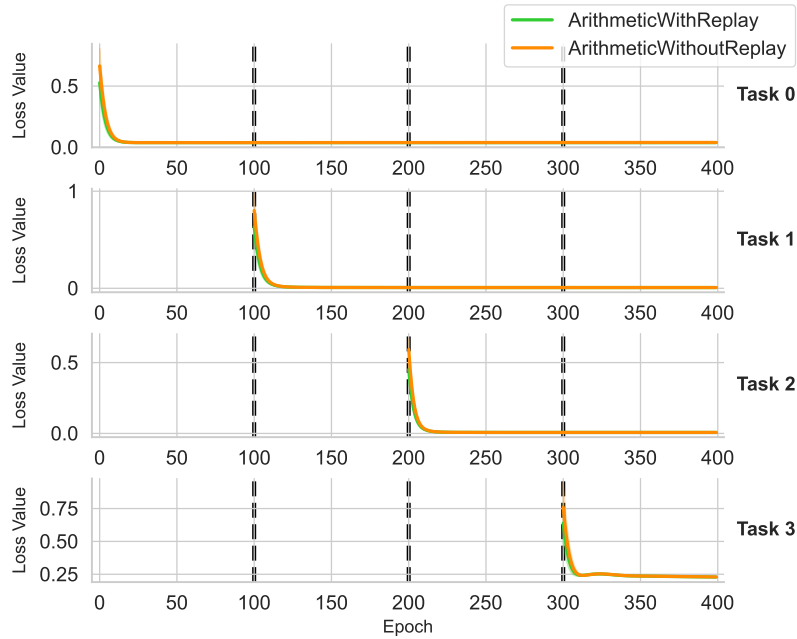# C   curves for forgetting experiments on arithmetic

Figure 6: Loss Curves of Arithmetic for each task

D   Task Hyper parameter setting

The choice of task generation hyper parameters impacts the experimental results. Hyper parameters that make the tasks too easy may lead to trivial performance differences, while excessively difficult tasks can hinder learning altogether. We thus report the choice of task generation hyper parameters we selected in the experiments.

D.1   Arithmetic

- **Range of Numbers:** 0 to 79

D.2   Tree

- **Number of Nodes:** 40
- **Maximum Children per Node:** 3
- **Minimum Children per Node:** 2

D.3   Graph

- **Number of Nodes:** 30
- **Maximum Edge Generation Probability:** 0.1
- **Minimum Edge Generation Probability:** 0.01

E Model Hyper parameters

E.1 ILP

- **Knowledge Base Settings:**
  - $[[8, 8, 8, 8]]$ – 8 predicates of arity 0 through 3, 1 layer
  - $[[8, 8, 8, 8], [8, 8, 8, 8]]$ – 8 predicates of arity 0 through 3, 2 layers
  - $[[8, 8, 8, 8], [8, 8, 8, 8], [8, 8, 8, 8]]$ – 8 predicates of arity 0 through 3, 3 layers
  - $[[8, 8, 8, 8], [8, 8, 8, 8], [8, 8, 8, 8], [8, 8, 8, 8]]$ – 8 predicates of arity 0 through 3, 4 layers
- **Task-Specific Module Setting:**
  - $[[8, 8, 8, 8]]$ – 8 predicates for arity 0 to 3 for composing the target predicate
- **Learning Rate:** 0.001

E.2 RL

- **Knowledge Base Settings:**
  - $[[4, 4, 4, 4]]$ – 4 predicates of arity 0 through 3, 1 layer
  - $[[4, 4, 4, 4], [4, 4, 4, 4]]$ – 4 predicates of arity 0 through 3, 2 layers
  - $[[4, 4, 4, 4], [4, 4, 4, 4], [4, 4, 4, 4]]$ – 4 predicates of arity 0 through 3, 3 layers
  - $[[4, 4, 4, 4], [4, 4, 4, 4], [4, 4, 4, 4], [4, 4, 4, 4]]$ – 4 predicates of arity 0 through 3, 4 layers
- **Task-Specific Module Setting:**
  - $[[8, 8, 8, 8]]$ – 8 predicates of arity 0 to 3 for composing the action predicates
- **Actor Learning Rate:** 0.003
- **Critic Learning Rate:** 0.003
- **Target Network Copy Rate:** 0.05
- **Actor Update Delay:** 0.05
- **Temperature for Generating Action Distribution:** 1.5
- **Activation for generating Action Logits:** tanh
- **Collected Buffer Size:** 50,000
- **Random Exploration Rate when Collecting Transitions:** 0.8

F Background for Logic Programming and ILP

This section includes some background information about *logic programming*, followed by its learning counterpart, *inductive logic programming*. ***Note that the material is primarily adapted from existing sources and can be readily obtained from other resources such as (Cropper and Dumančić 2022)***.

F.1 Logic Programming

Logic programming is a programming paradigm that execute based on formal logic sentences, rather than commands or functions typical of conventional programming. Knowledge about a domain is typically represented using logic facts and rules in first-order logic, with computation performed by iteratively applying these rules to deduce new facts. Here we include the definitions and concepts that are commonly used in logic programming.

Predicate A predicate p characterizes a property of an object or describes a relationship among multiple objects. It functions as a Boolean operator, returning either $True$ or $False$ based on its inputs. Consequently, a predicate can be represented as $p(t)$ in the case of a single-variable predicate or $p(t_1, t_2, ...)$ for predicates involving multiple variables.

Atoms An atom is formed by expressing a predicate along with its terms, which can be either constants or variables. It becomes grounded when all its inputs are replaced by specific constants (objects) within a domain, thereby establishing a definite relationship among these objects, and can be evaluated as $True$ or $False$. As the fundamental unit in logical expressions, it is aptly termed "atom".

Literal A literal is an atom or its negation, representing an atomic formula that contains no other logical connectives such as $\wedge$, $\vee$.

Clauses A clause is constructed using a finite set of literals and connectives. The most commonly used form of clause is the **definite clause**, which represents a disjunction of literals with exactly one positive literal included. It is expressed as

$$p_1 \vee \neg p_2 \vee \neg p_3 ... \vee \neg p_n$$

The truth value of the expression above is equivalent to that of an implication, allowing it to be rewritten in the form of an implication as follows:

$$p_1 \leftarrow p_2 \wedge p_3 ... \wedge p_n$$

The part to the left of the implication arrow is typically referred to as the **head** of the clause, while the part to the right is known as the **body**. A clause is read from right to left as the head is entailed by the conjunction of the body. A clause is **grounded** when all its literals are instantiated with constants from a domain. A ground clause without a body constitutes a **fact** about the domain, defined by the specific predicate and associated objects. A clause that includes variables is often treated as a rule about a domain.

Logic programming involves defining a set of clauses R including both rules and facts. The consequences of the set R is computed by repeatedly applying the rules in R until no more facts could be derived, where a convergence is considered to have been reached. We take the example from (Evans and Grefenstette 2018) to give an illustration on logic programming. Consider the program R as

$$edge(a, b) \quad edge(b, c) \quad edge(c, a)$$
$$connected(X, Y) \leftarrow edge(X, Y)$$
$$connected(X, Z) \leftarrow edge(X, Y) \wedge connected(Y, Z)$$

where $\{a, b, c, d\}$ represents the set of objects considered in the domain, while $\{X, Y, Z\}$ are variables in each predicate rule. Thus, $\{edge(a.b), edge(b, c), edge(c, d)\}$ are considered ground facts while the other clauses are considered logic rules that could be applied to derived new facts about a domain.

The computation of consequences of the program could be summarized as

$$C_{R,1} = \{edge(a, b), edge(b, c), edge(c, a)\}$$
$$C_{R,2} = C_{R,1} \cup \{connected(a, b), connected(b, c), connected(c, a)\}$$
$$C_{R,3} = C_{R,2} \cup \{connected(a, c), connected(b, a), connected(c, b)\}$$
$$C_{R,4} = C_{R,3} \cup \{connected(a, a), connected(b, b), connected(c, c)\}$$
$$C_{R,5} = C_{R,4} = con(R)$$

Thus, from the single-direction edge predicate and the rules defining the connection between two nodes, we deduce that all nodes are connected, including self-connections. $C_{R,5}$ constitutes the final conclusions of the program, as no additional facts are added beyond $C_{R,4}$.

F.2 Inductive Logic Programming

Inductive logic programming(ILP) aims for the opposite goal of logic programming: given a set of facts, it seeks to derive the rules that best explain those facts. We include several more definitions here to better explain the topic.

Background and Target Predicates We categorize the predicates involved in a problem into two sets: background predicates, which provide the foundational facts for logical deductions, and target predicates, which represent the conclusions derived from these deductions.

Extensional and Intentional Predicates In the context of clauses, extensional predicates never appear in the heads of clauses, whereas intentional predicates can. Background predicates, often the starting points for logical deduction, are typically extensional. In contrast, target predicates, which result from logical deductions, are usually intentional. The concept of intentional predicates is crucial, as ILP often requires inventing intermediary intentional predicates to facilitate program solving.

An inductive logic programming problem is defined by a tuple $\{B, E^+, E^-\}$ of ground atoms, where B specifies the background knowledge, which consists of ground atoms formed from the set of background predicates. E^+ and E^- denote the positive and negative targets, respectively, which are ground atoms formed from the set of target predicates.

The solution to an ILP involves finding a set of lifted rules U such that,

$$\forall e \in E^+, B \cup U \models e$$
$$\forall e \in E^-, B \cup U \not\models e$$

where $\models$ denotes logic entailment. More intuitively, this means that ground atoms from E^+ are included in the deduction consequences, while ground atoms from E^- are excluded.

We take an example from (Evans and Grefenstette 2018) to illustrate the idea. Suppose a ILP problem represented as

$$B = \{zero(0), succ(0,1), succ(1,2), ...\}$$
$$E^+ = \{even(0), even(2), even(4), ...\}$$
$$E^- = \{even(1), even(3), even(5), ...\}$$

where the predicates $zero()$ and $succ()$ serve as background predicates that facilitate the definition of the knowledge base, while the predicate $even()$ defines the learning target as the target predicate. A possible solution R for the ILP could be

$$even(X) \leftarrow zero(X)$$
$$even(X) \leftarrow even(Y) \wedge succ2(Y,X)$$
$$succ2(X,Y) \leftarrow succ(X,Z) \wedge succ(Z,Y)$$

An obvious deduction process would be

$$C_{R,1} = B$$
$$C_{R,2} = C_{R,1} \cup \{even(0), succ2(0,2), succ2(2,4), succ2(4,6), ...\}$$
$$C_{R,3} = C_{R,2} \cup \{even(2), even(4), even(6), ...\}$$
$$C_{R,4} = C_{R,2} = con(R)$$

Clearly, it is satisfied that $\forall e \in E^+, e \in con(R)$ and $\forall e \in E^-, e \notin con(R)$, thus the solution is accepted. The above example is not totally trivial, as it illustrates the concept of **predicate invention**: $succ2()$ is synthesised to facilitate logic deduction. To connect the example with the concepts defined in the beginning of the section, predicates $zero()$ and $succ()$ are extensional predicates as they never have to be deduced from the rules while $succ2()$ and $even()$ are intentional as they appear in the deduction heads of the rules.